PATENT APPLICATION

INSTRUCTION CACHE ASSOCIATIVE CROSSBAR SWITCH

Inventor: HOWARD G. SACHS
                   298 Bella Vista
                   Belvedere, California 94920
                   Citizenship: United States

Assignee: INTERGRAPH CORPORATION
                   One Madison Industrial Park
                   289 Dunlop Boulevard
                   Huntsville, Alabama 35824
                   Incorporation: Delaware

INSTRUCTION CACHE ASSOCIATIVE CROSSBAR SWITCH

BACKGROUND OF THE INVENTION

This invention relates to the architecture of computing systems, and in particular to an architecture in which individual instructions may be executed in parallel, as well as to methods and apparatus for accomplishing that.

A common goal in the design of computer architectures is to increase the speed of execution of a given set of instructions. One approach to increasing instruction execution rates is to issue more than one instruction per clock cycle, in other words, to issue instructions in parallel. This allows the instruction execution rate to exceed the clock rate. Computing systems that issue multiple independent instructions during each clock cycle must solve the problem of routing the individual instructions that are dispatched in parallel to their respective execution units. One mechanism used to achieve this parallel routing of instructions is generally called a "crossbar switch."

In present state of the art computers, e.g. the Digital Equipment Alpha, the Sun Microsystems SuperSparc, and the Intel Pentium, the crossbar switch is implemented as part of the instruction pipeline. In these machines the crossbar is placed between the instruction decode and instruction execute stages. This is because the conventional approach requires the instructions to be decoded before it is possible to determine the pipeline to which they should be dispatched. Unfortunately, decoding in this manner slows system speed and requires extra surface area on the integrated circuit upon which the processor is formed. These disadvantages are explained further below.

SUMMARY OF THE INVENTION

We have developed a computing system architecture that enables instructions to be routed to an appropriate pipeline more quickly, at lower power, and with simpler circuitry than previously possible. This invention places the crossbar switch earlier in the pipeline, making it a part of the initial instruction fetch operation. This allows the crossbar to be a part of the cache itself, rather than a stage in the instruction pipeline. It also allows the crossbar to take advantage of circuit design parameters that are typical of regular memory structures rather than random logic. Such advantages include: lower switching voltages (200 - 300 milliamps rather than 3 - 5 volts); more compact design, and higher switching speeds. In addition, if the crossbar is placed in the cache, the need for many sense amplifiers is eliminated, reducing the circuitry required in the system as a whole.

To implement the crossbar switch, the instructions coming from the cache, or otherwise arriving at the switch, must be tagged or otherwise associated with a pipeline identifier to direct the instructions to the appropriate pipeline for execution. In other words, pipeline dispatch information must be available at the crossbar switch at instruction fetch time, before conventional instruction decode has occurred. There are several ways this capability can be satisfied: In one embodiment this system includes a mechanism that routes each instruction in a set of instructions to be executed in parallel to an appropriate pipeline, as determined by a pipeline tag applied to each instruction during compilation, or placed in a separate identifying instruction that accompanies the original instruction. Alternately the pipeline affiliation can be determined after compilation at the time that instructions are fetched from memory into the cache, using a special predecoder unit.

Thus, in one implementation, this system includes a register or other means, for example, the memory cells

providing for storage of a line in the cache, for holding
instructions to be executed in parallel. Each instruction
has associated with it a pipeline identifier indicative of
the pipeline to which that instruction is to be issued. A
crossbar switch is provided which has a first set of
connectors coupled to receive the instructions, and a second
set of connectors coupled to the processing pipelines to
which the instructions are to be dispatched for execution.
Means are provided which are responsive to the pipeline
identifiers of the individual instructions in the group
supplied to the first set of connectors for routing those
individual instructions onto appropriate paths of the second
set of connectors, thereby supplying each instruction in the
group to be executed in parallel to the appropriate
pipeline.

In a preferred embodiment of this invention the
associative crossbar is implemented in the instruction
cache. By placing the crossbar in the cache all switching
is done at low signal levels (approximately 200 - 300
millivolts). Switching at these low levels is substantially
faster than switching at higher levels (5 volts) after the
sense amplifiers. The lower power also eliminates the need
for large driver circuits, and eliminates numerous sense
amplifiers. Additionally by implementing the crossbar in
the cache, the layout pitch of the crossbar lines matches
the pitch of the layout of the cache.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating a typical
environment for a preferred implementation of this
invention;

Figure 2 is a diagram illustrating the overall
structure of the instruction cache of Figure 1;

Figure 3 is a diagram illustrating one embodiment
of the associative crossbar;

Figure 4 is a diagram illustrating another
embodiment of the associative crossbar; and

Figure 5 is a diagram illustrating another embodiment of the associative crossbar.

DESCRIPTION OF THE SPECIFIC EMBODIMENTS

~~Figure 1 is a block diagram of a computer system~~ incorporating the associative crossbar switch according to the preferred embodiment of this invention. The following briefly describes the overall preferred system environment within which the crossbar is incorporated. For additional information about the system, see copending U.S. Patent Application Serial No. 08/147,860, filed Nov. 5, 1993, and entitled "Software Scheduled Superscaler Computer Architecture," which is incorporated by reference herein. Figure 1 illustrates the organization of the integrated circuit chips by which the computing system is formed. As depicted, the system includes a first integrated circuit 10 that includes a central processing unit, a floating point ~~unit, and an instruction cache.~~

In the preferred embodiment the instruction cache is a 16 kilobyte two-way set-associative 32 byte line cache. A set associative cache is one in which the lines (or blocks) can be placed only in a restricted set of locations. The line is first mapped into a set, but can be placed anywhere within that set. In a two-way set associative cache, two sets, or compartments, are provided, and each line can be placed in one compartment or the other.

The system also includes a data cache chip 20 that comprises a 32 kilobyte four-way set-associative 32 byte line cache. The third chip 30 of the system includes a predecoder, a cache controller, and a memory controller. The predecoder and instruction cache are explained further below. For the purposes of this invention, the CPU, FPU, data cache, cache controller and memory controller all may be considered of conventional design.

The communication paths among the chips are illustrated by arrows in Figure 1. As shown, the CPU/FPU and instruction cache chip communicates over a 32 bit wide

bus 12 with the predecoder chip 30. The asterisk is used to indicate that these communications are multiplexed so that a 64 bit word is communicated in two cycles. Chip 10 also receives information over 64 bit wide buses 14, 16 from the data cache 20, and supplies information to the data cache 20 over three 32 bit wide buses 18. The predecoder decodes a 32 bit instruction received from the secondary cache into a 64 bit word, and supplies that 64 bit word to the instruction cache on chip 10.

The cache controller on chip 30 is activated whenever a first level cache miss occurs. Then the cache controller either goes to main memory or to the secondary cache to fetch the needed information. In the preferred embodiment the secondary cache lines are 32 bytes and the cache has an 8 kilobyte page size.

The data cache chip 20 communicates with the cache controller chip 30 over another 32 bit wide bus. In addition, the cache controller chip 30 communicates over a 64 bit wide bus 32 with the DRAM memory, over a 128 bit wide bus 34 with a secondary cache, and over a 64 bit wide bus 36 to input/output devices.

As will be described further below, the system shown in Figure 1 includes multiple pipelines able to operate in parallel on separate instructions which are dispatched to these parallel pipelines simultaneously. In one embodiment the parallel instructions have been identified by the compiler and tagged with a pipeline identification tag indicative of the specific pipeline to which that instruction should be dispatched.

In this system, an arbitrary number of instructions can be executed in parallel. In one embodiment of this system the central processing unit includes eight functional units and is capable of executing eight instructions in parallel. These pipelines are designated using the digits 0 to 7. Also, for this explanation each instruction word is assumed to be 32 bits (4 bytes) long.

As briefly mentioned above, in the preferred embodiment the pipeline identifiers are associated with individual instructions in a set of instructions during compilation. In the preferred embodiment, this is achieved by compiling the instructions to be executed using a well-known compiler technology. During the compilation, the instructions are checked for data dependencies, dependence upon previous branch instructions, or other conditions that preclude their execution in parallel with other instructions. The result of the compilation is identification of a set or group of instructions which can be executed in parallel. In addition, in the preferred embodiment, the compiler determines the appropriate pipeline for execution of an individual instruction. This determination is essentially a determination of the type of instruction provided. For example, load instructions will be sent to the load pipeline, store instructions to the store pipeline, etc. The association of the instruction with the given pipeline can be achieved either by the compiler, or by later examination of the instruction itself, for example, during predecoding.

Referring again to Figure 1, in normal operation the CPU will execute instructions from the instruction cache according to well-known principles. On an instruction cache miss, however, a set of instructions containing the instruction missed is transferred from the main memory into the secondary cache and then into the primary instruction cache, or from the secondary cache to the primary instruction cache, where it occupies one line of the instruction cache memory. Because instructions are only executed out of the instruction cache, all instructions ultimately undergo the following procedure.

At the time a group of instructions is transferred into the instruction cache, the instruction words are predecoded by the predecoder 30. As part of the predecoding process, a multiple bit field prefix is added to each instruction based upon a tag added to the instruction by the

compiler. This prefix gives the explicit pipe number of the
pipeline to which that instruction will be routed. Thus, at
the time an instruction is supplied from the predecoder to
the instruction cache, each instruction will have a pipeline
5   identifier.
          It may be desirable to implement the system of
this invention on computer systems that already are in
existence and therefore have instruction structures that
have already been defined without available blank fields for
10   the pipeline information. In this case, in another
embodiment of this invention, the pipeline identifier
information is supplied on a different clock cycle, then
combined with the instructions in the cache or placed in a
separate smaller cache. Such an approach can be achieved by
15   adding a "no-op" instruction with fields that identify the
pipeline for execution of the instruction, or by supplying
the information relating to the parallel instructions in
another manner. It therefore should be appreciated that the
manner in which the instruction and pipeline identifier
20   arrives at the crossbar to be processed is somewhat
arbitrary. I use the word "associated" herein to designate
the concept that the pipeline identifiers are not required
to have a fixed relationship to the instruction words. That
is, the pipeline identifiers need not be embedded within the
25   instructions themselves by the compiler. Instead they may
arrive from another means, or on a different cycle.
          Figure 2 is a simplified diagram illustrating the
secondary cache, the predecoder, and the instruction cache.
This figure, as well as Figures 3, 4 and 5, are used to
30   explain the manner in which the instructions tagged with the
pipeline identifier are routed to their designated
instruction pipelines.
          In Figure 2, for illustration, assume that groups
of instructions to be executed in parallel are fetched in a
35   single transfer across a 256 bit (32 byte) wide path from a
secondary cache 50 into the predecoder 60. As explained
above, the predecoder prefixes the pipeline "P" field to the

instruction. After predecoding the resulting set of
instructions is transferred into the primary instruction
cache 70. At the same time, a tag is placed into the tag
field 74 for that line.

In the preferred embodiment the instruction cache
operates as a conventional physically-addressed instruction
cache. In the example depicted in Figure 2, the instruction
cache will contain 512 bit sets of instructions of eight
instructions each, organized in two compartments of
256 lines.

Address sources for the instruction cache arrive
at a multiplexer 80 that selects the next address to be
fetched. Because preferably instructions are always machine
words, the low order two address bits <1:0> of the 32 bit
address field supplied to multiplexer 80 are discarded.
These two bits designate byte and half-word boundaries. Of
the remaining 30 bits, the next three low order address bits
<4:2>, which designate a particular instruction word in the
set, are sent directly via bus 81 to the associative
crossbar. The next low eight address bits <12:5> are
supplied over bus 82 to the instruction cache 70 where they
are used to select one of the 256 lines in the instruction
cache. Finally, the remaining 19 bits of the virtual
address <31:13> are sent to the translation lookaside buffer
(TLB) 90. The TLB translates these bits into the high
19 bits of the physical address. The TLB then supplies them
over bus 84 to the instruction cache. In the cache they are
compared with the tag of the selected line, to determine if
there is a "hit" or a "miss" in the instruction cache.

If there is a hit in the instruction cache,
indicating that the addressed instruction is present in the
cache, then the selected set of instructions is transferred
across the 512 bit wide bus 73 into the associative crossbar
100. The associative crossbar 100 then dispatches the
addressed instructions to the appropriate pipelines over
buses 110, 111, ..., 117. Preferably the bit lines from the
memory cells storing the bits of the instruction are

themselves coupled to the associative crossbar. This
eliminates the need for numerous sense amplifiers, and
allows the crossbar to operate on the lower voltage swing
information from the cache line directly, without the

5   normally intervening driver circuitry to slow system
operation.

Figure 3 illustrates in more detail one embodiment
of the associative crossbar. A 512 bit wide register 130,
which represents the memory cells in a line of the cache (or

10   can be a physically separate register), contains at least
the set of instructions capable of being issued. For the
purposes of illustration, register 130 is shown as
containing up to eight instruction words W0 to W7. Using
means described in the copending application referred to

15   above, the instructions have been sorted into groups for
parallel execution. For illustration here, assume the
instructions in Group 1 are to be dispatched to pipelines 1,
2 and 3; the instructions in Group 2 to pipelines 1, 3 and
6; and the instructions in Group 3 to pipelines 1 and 6.

20   The decoder select signal enables only the appropriate set
of instructions to be executed in parallel, essentially
allowing register 130 to contain more than just one set of
instructions. Of course, by only using register 130 only
for one set of parallel instructions at a time, the decoder

25   select signal is not needed.

As shown in Figure 3, the crossbar switch itself
consists of two sets of crossing pathways. In the
horizontal direction are the pipeline pathways 180, 181,
..., 187. In the vertical direction are the instruction

30   word paths, 190, 191, ..., 197. Each of these pipeline and
instruction pathways is themselves a bus for transferring
the instruction word. Each horizontal pipeline pathway is
coupled to a pipeline execution unit 200, 201, 202, ...,
207. Each of the vertical instruction word pathways 190,

35   191, ..., 197 is coupled to an appropriate portion of
register or cache line 130.

The decoders 170, 171, ..., 177 associated with each instruction word pathway receive the 4 bit pipeline code from the instruction. Each decoder, for example decoder 170, provides eight 1 bit control lines as output. One of these control lines is associated with each pipeline pathway crossing of that instruction word pathway. Selection of a decoder as described with reference to Figure 3 activates the output bit control line corresponding to that input-pipe number. This signals the crossbar to close the switch between the word path associated with that decoder and the pipe path selected by that bit line. Establishing the cross connection between these two pathways causes a selected instruction word to flow into the selected pipeline. For example, decoder 173 has received the pipeline bits for word W3. Word W3 has associated with it pipeline path 1. The pipeline path 1 bits are decoded to activate switch 213 to supply instruction word W3 to pipeline execution unit 201 over pipeline path 181. In a similar manner, the identification of pipeline path 3 for decoder D4 activates switch 234 to supply instruction word W4 to pipeline path 3. Finally, the identification of pipeline 6 for word W5 in decoder D5 activates switch 265 to transfer instruction word W5 to pipeline execution unit 206 over pipeline pathway 186. Thus, instructions W3, W4 and W5 are executed by pipes 201, 203 and 206, respectively.

The pipeline processing units 200, 201, ..., 207 shown in Figure 3 can carry out desired operations. In a preferred embodiment of the invention, each of the eight pipelines first includes a sense amplifier to detect the state of the signals on the bit lines from the crossbar. In one embodiment the pipelines include first and second arithmetic logic units; first and second floating point units; first and second load units; a store unit and a control unit. The particular pipeline to which a given instruction word is dispatched will depend upon hardware constraints as well as data dependencies.

Figure 4 is a diagram illustrating another embodiment of the associative crossbar. In Figure 4 nine pipelines 0 - 8 are shown coupled to the crossbar. The decode select is used to enable a subset of the instructions in the register 130 for execution just as in the system of Figure 3.

The execution ports that connect to the pipelines specified by the pipeline identification bits of the enabled instructions are then selected to multiplex out the appropriate instructions from the contents of the register. If one or more of the pipelines is not ready to receive a new instruction, a set of hold latches at the output of the execution ports prevents any of the enabled instructions from issuing until the "busy" pipeline is free. Otherwise the instructions pass transparently through the hold latches into their respective pipelines. Accompanying the output of each port is a "port valid" signal that indicates whether the port has valid information to issue to the hold latch.

Figure 5 illustrates an alternate embodiment for the invention where pipeline tags are not included with the instruction, but are supplied separately, or where the cache line itself is used as the register for the crossbar. In these situations, the pipeline tags may be placed into a high speed separate cache memory 200. The output from this memory can then control the crossbar in the same manner as described in conjunction with Figure 3. This approach eliminates the need for sense amplifiers between the instruction cache and the crossbar. This enables the crossbar to switch very low voltage signals more quickly than higher level signals, and the need for hundreds of sense amplifiers is eliminated. To provide a higher level signal for control of the crossbar, sense amplifier 205 is placed between the pipeline tag cache 200 and the crossbar 100. Because the pipeline tag cache is a relatively small memory, however, it can operate more quickly than the instruction cache memory, and the tags therefore are available in time to control the crossbar despite the sense

amplifier between the cache 200 and the crossbar 100. Once the switching occurs in the crossbar, then the signals are amplified by sense amplifiers 210 before being supplied to the various pipelines for execution.

The architecture described above provides many unique advantages to a system using this crossbar. The crossbar described is extremely flexible, enabling instructions to be executed sequentially or in parallel, depending entirely upon the "intelligence" of the compiler. Importantly, the associative crossbar relies upon the content of the message being decoded, not upon an external control circuit acting independently of the instructions being executed. In essence, the associative crossbar is self directed.

Another important advantage of this system is that it allows for more intelligent compilers. Two instructions which appear to a hardware decoder (such as in the prior art described above) to be dependent upon each other can be determined by the compiler not to be interdependent. For example, a hardware decoder would not permit two instructions R1 + R2 = R3 and R3 + R5 = R6 to be executed in parallel. A compiler, however, can be "intelligent" enough to determine that the second R3 is a previous value of R3, not the one calculated by R1 + R2, and therefore allow both instructions to issue at the same time. This allows the software to be more flexible and faster.

Although the foregoing has been a description of the preferred embodiment of the invention, it will be apparent to those of skill in the art the numerous modifications and variations may be made to the invention without departing from the scope as described herein. For example, arbitrary numbers of pipelines, arbitrary numbers of decoders, and different architectures may be employed, yet rely upon the system we have developed.